

Article

A Real-Time Tree Crown Detection Approach for Large-Scale Remote Sensing Images on FPGAs

Weijia Li ^{1,2,†} , Conghui He ^{3,4,†}, Haohuan Fu ^{1,2,*}, Juepeng Zheng ^{1,2,5}, Runmin Dong ^{1,2}, Maocai Xia ^{1,2}, Le Yu ^{1,2}  and Wayne Luk ⁶

¹ Ministry of Education Key Laboratory for Earth System Modeling, Department of Earth System Science, Tsinghua University, Beijing 100084, China; liwj14@mails.tsinghua.edu.cn (W.L.); 1351177@tongji.edu.cn (J.Z.); drm17@mails.tsinghua.edu.cn (R.D.); xiamc16@mails.tsinghua.edu.cn (M.X.); leyu@tsinghua.edu.cn (L.Y.)

² Joint Center for Global Change Studies (JCGCS), Beijing 100084, China

³ Department of Computer Science, Tsinghua University, Beijing 100084, China; heconghui@sensetime.com

⁴ SenseTime Group Limited, Shenzhen 518000, China

⁵ College of Surveying and Geo-Informatics, Tongji University, Shanghai 200092, China

⁶ Department of Computing, Imperial College London, London SW7 2RH, UK; w.luk@imperial.ac.uk

* Correspondence: haohuan@tsinghua.edu.cn; Tel.: +86-010-62798365

† These authors contributed equally to this work.

Received: 28 March 2019; Accepted: 22 April 2019; Published: 30 April 2019



Abstract: The on-board real-time tree crown detection from high-resolution remote sensing images is beneficial for avoiding the delay between data acquisition and processing, reducing the quantity of data transmission from the satellite to the ground, monitoring the growing condition of individual trees, and discovering the damage of trees as early as possible, etc. Existing high performance platform based tree crown detection studies either focus on processing images in a small size or suffer from high power consumption or slow processing speed. In this paper, we propose the first FPGA-based real-time tree crown detection approach for large-scale satellite images. A pipelined-friendly and resource-economic tree crown detection algorithm (PF-TCD) is designed through reconstructing and modifying the workflow of the original algorithm into three computational kernels on FPGAs. Compared with the well-optimized software implementation of the original algorithm on an Intel 12-core CPU, our proposed PF-TCD obtains the speedup of 18.75 times for a satellite image with a size of $12,188 \times 12,576$ pixels without reducing the detection accuracy. The image processing time for the large-scale remote sensing image is only 0.33 s, which satisfies the requirements of the on-board real-time data processing on satellites.

Keywords: tree crown detection; high-resolution satellite images; field-programmable gate array (FPGA); real-time processing

1. Introduction

The automatic tree crown detection from high-resolution remote sensing images has been an important research topic with wide attention for decades [1,2]. The distribution and the number of trees in a plantation area are valuable information for predicting the yield of trees' fruitage, understanding the growing situation or survival rate of trees after plantation, and planning the irrigation or fertilization, etc. [3,4]. Owing to the rapid development of satellite, Unmanned Aerial Vehicle (UAV), and airborne remote sensing technology, the large-scale and high-resolution remote sensing images become increasingly abundant [5,6]. Various approaches for automatic tree crown detection have been proposed in many remote sensing studies.

In general, existing methods for automatic tree crown detection can be summarized into the following three categories. The first category is traditional image processing-based methods [7,8], more specifically, the image binarization [9], the template matching [10], the local maximum filter based methods [2], etc. The second category is traditional machine learning based methods, including maximum likelihood [11], support vector machine [12], extreme learning machine [13], random forest [14], etc. The third category is deep learning based methods, which are based on the convolutional neural networks [4,15–17]. Although supervised machine learning and deep learning based methods usually achieve relatively high accuracy, these methods require a large number of manually labeled training samples [14,18], which are still very limited for tree crown detection in large-scale areas. For traditional image processing based methods, the local maximum filter based method often achieves similar or higher detection accuracy compared with other traditional methods (e.g., template matching and binarization based methods) [4,9]. Moreover, no manually labeled dataset for tree crown detection is required for this method. For these reasons, we select the most widely used local maximum filter based method (proposed in [2]) as the prototype of the tree crown detection approach proposed in this study.

Due to the increasing quantity of remote sensing data and the growing speed of data acquisitions, many efforts have been made towards the on-board remote sensing data processing, which is not only aimed at avoiding the delay between data acquisition and data processing but also reducing the quantity of data transmission from the satellite to the ground [19–21]. Among various high performance platforms (e.g., multi-core processors [22], many-core processors [23], Graphics Processing Units (GPUs) [24]), the Field Programmable Gate Arrays (FPGAs) have become one of the most popular platforms for on-board remote sensing data processing for the following reasons. First, FPGAs have relatively smaller size and weight compared with clusters of computers, multi-core processors and many-core processors, etc. [25]. The second advantage is that FPGAs require significantly lower power consumption compared with GPUs, which is a great benefit considering the limited power budget of the satellites [26]. Moreover, different from ASICs, FPGAs have the inherent ability to be reprogrammed when their functionality needs to be updated during the lifetime of the satellite [27]. Owing to above reasons, FPGA-based platforms have been broadly used for accelerating many on-board data processing tasks, including hyperspectral image processing (e.g., classification, anomaly detection, unmixing, etc.) [28,29], cloud detection [30], vehicle detection [31], tree crown detection [32], etc.

Many of the FPGA-based on-board data processing studies are aimed at hyperspectral image processing. For instance, González et al. [25] developed an algorithm for automatically detecting anomaly targets in remotely sensed hyperspectral images on FPGA, achieving 4.6 times speedup for a 614×512 image over the serial implementation on CPU. Wang et al. [28] presented a scalable dataflow accelerator on FPGA for real-time SVM classification of hyperspectral images, achieving 14 times speedup for a 145×145 image over the implementation on a 12-core CPU. Moreover, in some studies, the FPGA has been applied to real-time on-board object detection for high resolution remote sensing images. For instance, Shan et al. [30] proposed an optimized architecture for real-time automatic cloud detection using the spectrum and texture analysis combination (STAC) approach, obtaining 3 times speedup compared with the standard STAC approach. Tang et al. [31] proposed an FPGA-based moving target detection system for UAV in real-time, obtaining 2.28 times speedup over the serial CPU for a 640×480 image. These existing FPGA-based studies often focus on on-board remote sensing image processing in a small scale.

In recent years, there have been several studies about accelerating tree crown detection algorithms on high performance platforms, which contribute to monitoring the growing condition of individual trees in real time and discovering the abnormal situations (e.g., the damage or cutting down of trees) as early as possible. The on-board tree crown detection from satellite images also enables that only a small quantity of tree crowns' coordinates is required to be transmitted from the satellite to the ground instead of a large quantity of remote sensing images. Duncanson et al. [33] developed an efficient tree crown delineation algorithm on 32-core CPUs. Although the proposed method is nearly

32 times faster than the serial implementation, it still takes 90 and 30 min to process two LiDAR datasets. Jiang et al. [34] developed a GPU-accelerated tree detection method for UAV images using a scale-space filtering method. It achieves 44.77 times and 28.54 times speedups for two testing images with a size of 4000×3000 pixels. The processing time of each image is less than 1 s, which satisfies the real-time processing requirement. However, the high power consumption of GPU makes it difficult to satisfy the limited power budget of satellites for on-board data processing [26]. Chen et al. [32] developed a parallel processing approach of the multi-level morphological active contour algorithm for tree crown detection on FPGA. The experimental results indicate that the proposed architecture could provide around 31% acceleration for an image with only 454×424 pixels compared with the simple implementation in Matlab. To the best of our knowledge, there is no related research about the tree crown detection algorithm used in our study (proposed in [2]) on FPGAs or any other high performance platforms.

In this paper, we design and implement an FPGA-based approach for automatic tree crown detection from large-scale remote sensing images, which satisfies the power, latency and performance requirements of the practical scenarios of on-board data processing. It is the first high performance design of the original tree crown detection algorithm used in this study (proposed in [2]). As one of the most classical tree crown detection algorithms for optical remote sensing images, the algorithm proposed in [2] has been widely used and cited in over three hundreds studies for decades. Experiment results show that our proposed design can provide a high performance solution for processing a large-scale remote sensing image of $12,188 \times 12,576$ pixels in 0.33 s, archiving 18.7-times speedup over the well-optimized software implementation on 12-core CPU. Our main contributions are summarized as follows.

- The direct mapping of the original tree crown detection algorithm to FPGAs results in high resource utilization and each pixel requires to be streamed for multiple times. Through reconstructing and modifying the original workflow into three computational kernels on FPGAs, we design a pipelined-friendly tree crown detection approach (PF-TCD) so that each pixel of the image can be streamed for only once.
- Through optimizing and adjusting the local maximum filtering, transect sampling, and minimum distance filtering algorithms of the tree crown detection approach, the utilization of different resources (look-up tables (LUTs), flip-flops (FFs), digital signal processor blocks (DSPs), and block random access memory (BRAMs)) is reduced significantly and well balanced, avoiding any of them becoming the performance bottleneck.
- We propose a complete FPGA-based framework for processing the large-scale remote sensing image in real time, which provides a high performance solution for tree crown detection from the raw remote sensing images to the final detection results. The proposed framework can process the satellite image of $12,188 \times 12,576$ pixels in 0.33 s, achieving 18.7-times speedup over the well-optimized software implementation on 12-core CPU.

The rest of the paper is organized as follows. Section 2 describes the workflow of the original tree crown detection algorithm. Section 3 introduces the study area and data of this research. Section 4 presents the overall framework of the tree crown detection approach on FPGAs and the algorithm design of the three computational kernels. Section 5 analyzes the experimental results of our proposed approach. Section 6 discusses the performance of our proposed approach and other related studies. Section 7 presents the main conclusions and the future work of this study.

2. Background

The original tree crown detection algorithm developed in this study is based on the method proposed in [2], which has been one of the most widely used tree crown detection algorithm for optical remote sensing images for decades. Figure 1 shows the overall process of this algorithm, including image pre-processing, non-overlapping local maximum filtering, transect sampling,

circular-window-based local maximum filtering, and minimum distance filtering. The details of these phases are described from Sections 2.1–2.3.

2.1. Image Pre-Processing and Non-Overlapping Local Maximum Filtering

To enhance the distinctiveness of tree crown apex, an image pre-processing method is undertaken for each pixel to calculate an index from its original band values that can best identify the tree crown from other types of pixels. Several calculation methods can be applied to obtain the index. Based on the existing studies [35] and our experiment tests, we find that the best index for remote sensing images with only Red, Green and Blue bands and those with Red, Green, Blue and Near Infrared (NIR) bands can be obtained according to Equation (1), where n_{band} is the number of bands of the input image; R_i , G_i and N_i are the values of Red, Green and NIR band of pixel i , respectively; P_i is the calculated index of pixel i . After this phase, the multi-band remote sensing image will be converted into a single-band image, as shown in Figure 1a,b.

$$P_i = \begin{cases} (G_i - R_i) / (G_i + R_i), & n_{band} = 3 \\ |N_i - R_i|, & n_{band} = 4 \end{cases} \quad (1)$$

After image pre-processing, a non-overlapping local maximum filter is applied to the single-band image to find the initial coordinates of potential tree crowns. The window width of each local maximum filter (denoted by w) should not be larger than the average diameter of each tree crown in the image. The moving distance in each step is equal to the window width for the non-overlapping local maximum filter. For the j_{th} window (denoted by W_j) in the image with a size of $W \times W$ pixels, the calculation process can be summarized according to Equations (2)–(5), where P_k is the value of the k_{th} pixel in a window; $M(W_j)$ is the maximum pixel value of W_j ; $I(W_j)$ is the index of $M(W_j)$ relative to the window W_j ; $X(W_j)$ and $Y(W_j)$ are the X and Y coordinates of $M(W_j)$ relative to the whole image. The number of windows in the whole image is denoted by N . The result obtained after this phase is shown in Figure 1b, in which each black point denotes the local maximum of each window.

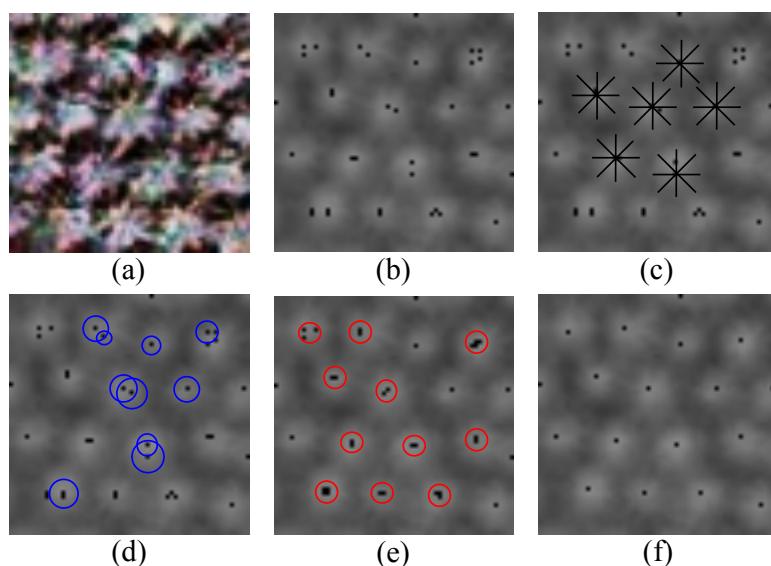


Figure 1. The overall process of the original tree crown detection algorithm. (a) The input image; (b) Image pre-processing and non-overlapping local maximum filtering; (c) Transect sampling; (d) Circular-window-based local maximum filtering; (e) Minimum distance filtering; (f) The output results of the tree crown detection algorithm.

$$M(W_j) = \max\{P_k : k \in W_j\} \quad (2)$$

$$I(W_j) = \operatorname{argmax}\{P_k : k \in W_j\} \quad (3)$$

$$X(W_j) = (j\%(W/w)) \times w + I(W_j)\%w \quad (4)$$

$$Y(W_j) = (j/(W/w)) \times w + I(W_j)/w \quad (5)$$

2.2. Transect Sampling and Circular-Window-Based Local Maximum Filtering

In this phase, we adjust the initial coordinates of potential tree apexes obtained in the former phase to new positions based on the transect sampling strategy and the circular-window-based local maximum filtering strategy, as shown in Figure 1c,d. In the phase of transect sampling, we find the radius with the maximum change of pixel value for each transect and calculate the average radius of all transects. Figure 1c shows some examples of the transects using the black lines.

The process of the transect sampling strategy is described in Algorithm 1. N_w , N_t and n_t denote the number of windows, the number of transects, and the number of pixels in a transect, respectively. Δr is the change of radius in each step. $x_{q'}$, $y_{q'}$ and $I_{q'}$ are the x coordinate, the y coordinate, and the index of the current pixel (q) or the next pixel ($q + 1$), all of which are relative to the whole image. C_q is the change of pixel value between two neighboring pixels. R_p is the radius with the maximum change of pixel value for the p_{th} transect. R_j is the average radius of all transects.

In the phase of circular-window-based local maximum filtering, R_j is used as the radius of the circular window. For each local maximum's coordinate obtained in Section 2.1 (denoted by the black points in Figure 1d), we find the new local maximum's coordinate within the corresponding circular window (denoted by the blue circles in Figure 1d). The process of circular-window-based local maximum filtering is similar to the one of non-overlapping local maximum filtering described in Section 2.1. The updated tree apexes (denoted by black points in Figure 1e) might be out of the former windows used in non-overlapping local maximum filtering.

Algorithm 1 Transect sampling

```

for all  $j \in \{0, 1, \dots, N_w - 1\}$  do
    for all  $p \in \{0, 1, \dots, N_t - 1\}$  do
        for all  $q \in \{0, 1, \dots, n_t - 1\}$  do
            for all  $q' \in \{q, q + 1\}$  do
                 $x_{q'} = X(W_j) + q' \times \Delta r \times \sin\left(\frac{2\pi}{n_t} \times p\right)$ 
                 $y_{q'} = Y(W_j) - q' \times \Delta r \times \cos\left(\frac{2\pi}{n_t} \times p\right)$ 
                 $I_{q'} = x_{q'} + y_{q'} \times W$ 
            end for
        end for
         $C_q = P(I_{q+1}) - P(I_q)$ 
    end for
     $R_p = \operatorname{argmax}\{C_q : q \in \{0, 1, \dots, n_t - 1\}\} \times \Delta r$ 
end for
     $R_j = \frac{\sum_{p=0}^{N_t-1} R_p}{N_t}$ 
end for

```

2.3. Minimum Distance Filtering

In the phase of minimum distance filtering, we average a group of local maximum's coordinates into one coordinate if the distances between these coordinates are smaller than the allowed physical structure of a tree (the black points within the red circles in Figure 1e). The process of this phase is described in Algorithm 2. X_{sum} and Y_{sum} are the sum of coordinates of a group of local maxima. N_{merge} is the number of local maxima in a group. $X'(W_j)$ and $Y'(W_j)$ are the local maximum's coordinates of

the j_{th} window obtained in Section 2.2. S_j and S_{jj} are the status of the local maximum's coordinate of the j_{th} and the jj_{th} windows, in which the False indicates the coordinate has been removed while the Truth indicates the coordinate still exists. d_{jj} is the Euclidean distance between the local maximum's coordinates of the j_{th} and the jj_{th} windows. d_{min} is the minimum allowed distance between two tree apexes. X_{avg} and Y_{avg} are the average coordinates of a group of local maxima. After this phase, we can obtain the final tree crown detection results. The final tree crown coordinates are denoted by the black points in Figure 1f.

Algorithm 2 Minimum distance filtering

```

for all  $j \in \{0, 1, \dots, N_w - 1\}$  do
  if  $S_j == True$  then
     $X_{sum} = 0, Y_{sum} = 0, N_{merge} = 0$ 
    for all  $jj \in \{0, 1, \dots, N_w - 1\}$  do
       $d_{jj} = dist((X'(W_j), Y'(W_j)), (X'(W_{jj}), Y'(W_{jj})))$ 
      if  $(S_{jj} == True) \& (d_{jj} < d_{min})$  then
         $X_{sum} += X'(W_{jj}), Y_{sum} += Y'(W_{jj}), N_{merge} += 1, S_{jj} = False$ 
      end if
    end for
     $X_{avg} = \frac{X_{sum}}{N_{merge}}, Y_{avg} = \frac{Y_{sum}}{N_{merge}}$ 
  end if
end for
  
```

3. Data

The remote sensing data used in this research is a large-scale Quickbird satellite image collected from the south of Malaysia on 21 November 2006, as shown in Figure 2. There are different land cover types in this study area, e.g., oil palm plantation areas, grassland, forests, buildings, bare land, etc. The Quickbird satellite image has four spectral bands (Red, Green, Blue, and NIR) with a size of $12,188 \times 12,576$ pixels and a spatial resolution of 0.6 meters. Six regions are selected from the whole study area (denoted by red squares in Figure 2, each with a size of 600×600 pixels and various land cover types) for evaluating the detection accuracy of our proposed approach on FPGAs.

In our study area, most of the regions are covered by the oil palm trees, which are one of the most vital economic crops in Malaysia and other tropical countries [36]. The palm oil produced by the oil palm trees is the most consumed vegetable oil all over the world [16]. The automatic detection of oil palm trees from the high-resolution satellite image is an important research issue for understanding the growing situation of oil palms, improving the resource utilization, and other essential aspects of the precision agriculture of oil palms [3]. The crown diameter of a mature oil palm is about ten meters (i.e., 15–17 pixels in the Quickbird image).

Moreover, many oil palm mapping studies have published regional or worldwide oil palm plantation maps [36,37], which can be combined with the tree crown detection algorithm for optimizing the oil palm detection results. In this study, We select the 0.6-meter-resolution oil palm plantation maps (https://github.com/dongrunmin/oil_palm_data/blob/master/segmentation_map) published in [38] (with the same spatial resolution as the Quickbird image used in this study) for further optimizing the oil palm detection results obtained from the tree crown detection algorithm.

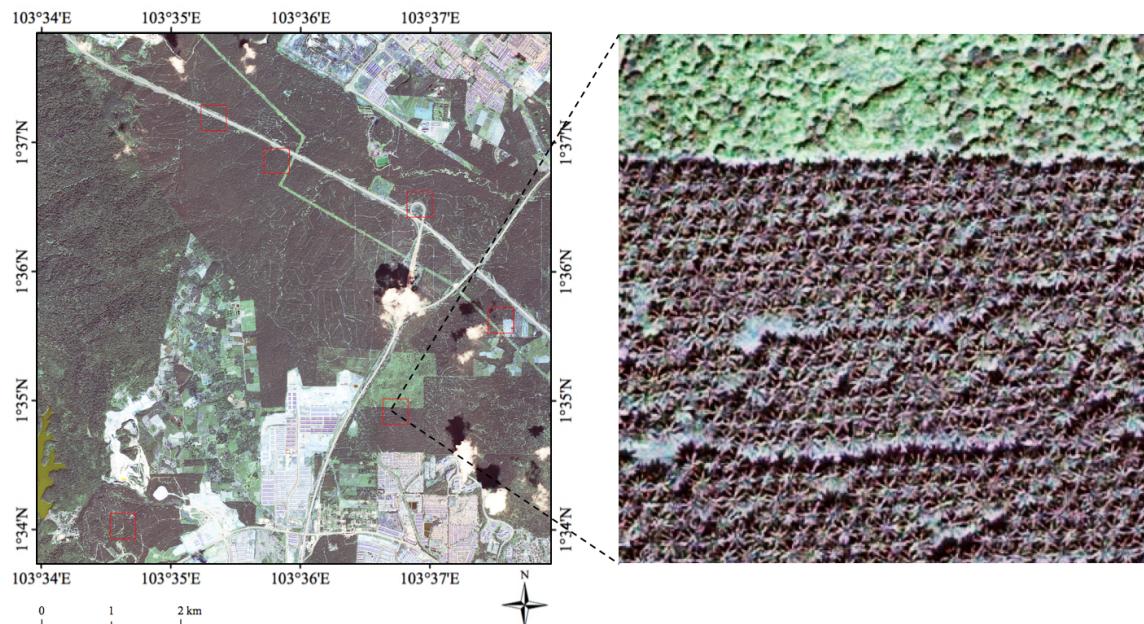


Figure 2. The Quickbird satellite image of the whole study area (left) and a local area of accuracy evaluation regions (right). The red squares denote the selected regions for evaluating the detection accuracy.

4. Methods

In this section, we introduce the design of our proposed PF-TCD algorithm on FPGAs. The overall framework of PF-TCD algorithm will be described in Section 4.1. The three computational kernels will be described from Sections 4.2–4.4.

4.1. Overall Framework of Tree Crown Detection for Large-Scale Remote Sensing Images on FPGAs

In this paper, we propose a framework of tree crown detection that uses reconfigurable platforms such as FPGAs to process the large-scale remote sensing images in real-time on satellites. The framework is shown in Figure 3. We use FPGAs as the ideal processing units as they not only meet the limited power and weight budget of satellites but also provide high performance and real-time computation. As FPGAs are much more efficient than general processing units, we offload as many tasks as possible to FPGAs in order to squeeze every bit of performance, including image pre-processing, non-overlapping local maximum filtering, transect sampling, circular-window-based local maximum filtering and minimum distance filtering. In order to further improve the performance, we propose a complete set of customized mechanisms on FPGAs for the tree crown detection algorithm from the raw remote sensing images to the final detection results.

The direct mapping of original tree crown detection algorithm to FPGAs results in a very naive design, which suffers from high resource utilization as well as requiring each pixel to be streamed for multiple times. In order to solve these problems, we propose a pipeline-friendly tree crown detection algorithm (PF-TCD) by reconstructing and modifying the original workflow. In our proposed PF-TCD algorithm, we only use 1/3 of the total resources and each pixel can be streamed for only once, leading to a pipeline-friendly and resource-economic design of the tree crown detection algorithm. The proposed PF-TCD algorithm consists of three computational kernels: (1) Transect sampling radius calculation and local maximum filtering; (2) Transect sampling radius based local maximum filtering; (3) Minimum distance filtering. The details of each kernel will be described from Sections 4.2–4.4.

To further reduce and balance the utilization of different resources, the large-scale image is partitioned into smaller image blocks. Multiple image blocks are then streamed into three PF-TCD kernels and processed in parallel. As image blocks contain overlaps with their neighbours to guarantee

that each tree crown can be completely included in at least one image block, smaller image blocks will increase the total number of image blocks and duplicated pixels. On the other hand, smaller image blocks often require fewer resource utilization so that more image blocks can be processed in parallel. So we adjust the size of the image block to maximize the overall performance. As multiple image blocks are processed in parallel, the bandwidth between the PCIe and FPGAs becomes the bottleneck. In order to avoid this issue, we first load the raw images to the off-chip memory on FPGAs before streaming the images into PF-TCD algorithm.

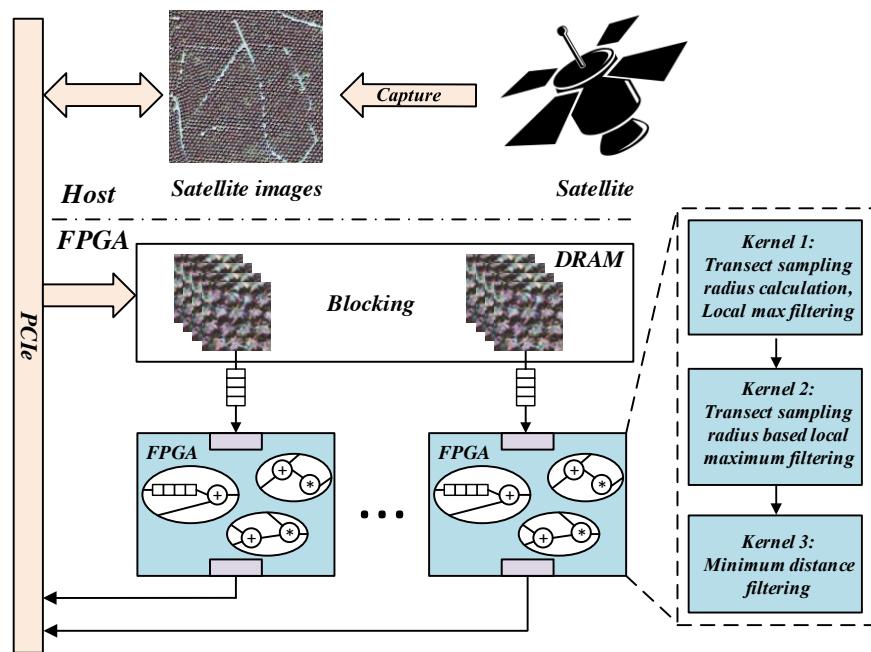


Figure 3. The overall framework of the tree crown detection for large-scale satellite images on FPGAs.

4.2. Kernel 1: Transect Sampling Radius Calculation and Local Maximum Filtering

In the original tree crown detection algorithm proposed in [2], the non-overlapping local maximum filtering is applied to the whole image in order to find the local maximum's coordinates for each window. Then the transect sampling scheme is applied to each local maximum in order to find the radius with the maximum change of pixel value for each transect. It is a challenging task to search for the local maximum's coordinates of each window from the large-scale remote sensing images on FPGAs, resulting in several problems if we directly map the original algorithm to FPGA platform. First, to search for the local maximum's coordinates of a $w \times w$ window in each cycle, we need to access the streams of all pixel values in the window, which requires a large amount of on-chip buffer with a size of $W \times w + w$ (W denotes the width of the image). Second, it takes $w \times w$ cycles to slide through a $w \times w$ window during the local maximum filtering phase. In $w \times w$ cycles, only the result of one cycle is useful while the results of the other ($w \times w - 1$) cycles are ignored. Third, it requires a lot of resources for the utilization of mathematical functions (e.g., \sin , \cos , and $\sqrt{}$) during the transect sampling phase.

In our proposed PF-TCD algorithm, we design and implement a set of strategies to address the problems mentioned above. First, a carefully tuned blocking scheme is used to reduce the buffer size. Second, we avoid the utilization of \sin and \cos functions by choosing special angles such as 45° and 90° , which also generates correct results. We approximate the $\sqrt{}$ function using piece-wise linear function inspired from [39]. Moreover, we design a search-in-time strategy in Kernel 1 to replace the search-in-space scheme used in the original algorithm in order to increase the data utilization and decrease the resource usage. Details of the search-in-time strategy in Kernel 1 are introduced as follows.

Figure 4 shows the basic process of Kernel 1. We modify the processing workflow of the original algorithm. In each cycle, the first step is to calculate the average radius of all transects for the current input stream, and the second step is to update the local maximum value M , its X and Y coordinates, and the transect sampling radius R of its corresponding window. The above mentioned problems can be effectively solved in our proposed design of Kernel 1.

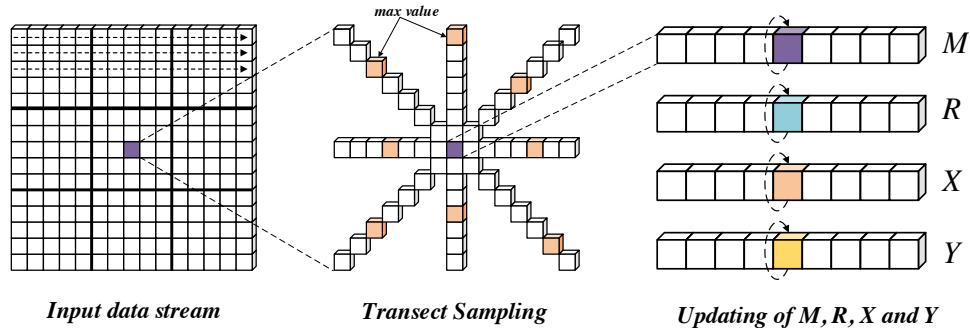


Figure 4. The basic process of Kernel 1.

In the first step of Kernel 1, the moving stride of each transect direction (S_p) can be defined as (6), where p is numbered in clockwise direction. The change of pixel value ($C_p(i)$) between the current stream (P_i) and the stream in the p_{th} transect direction (P_{i+S_p}) can be defined as Equation (7). For each pixel in the p_{th} transect direction, the change of pixel value between two neighbouring pixels ($C_p(q)$) can be defined as Equation (8) and the corresponding radius ($r_p(q)$) can be defined as Equation (9). The radius with the maximum change of pixel value can be defined as Equation (10). The average radius of all transects for $P(i)$ can be defined as Equation (11).

$$S_p = \{-W, -W + 1, 1, W + 1, W, W - 1, -1, -W - 1\}, p \in \{0, 1, 2, \dots, N_t - 1\} \quad (6)$$

$$C_p(i) = P_{i+S_p} - P_i \quad (7)$$

$$C_p(q) = P(i + (q + 1) \times S_p) - P(i + q \times S_p), q \in \{0, 1, \dots, n_t - 1\} \quad (8)$$

$$r_p(q) = \begin{cases} q + 1, & q \in \{0, 2, 4, 6\} \\ (q + 1) \times 1.41, & q \in \{1, 3, 5, 7\} \end{cases} \quad (9)$$

$$R_p = r_p(\text{argmax}\{C_p(q), q \in \{0, 1, \dots, n_t - 1\}\}) \quad (10)$$

$$R_i = \frac{\sum_{p=0}^{N_t-1} R_p}{N_t} \quad (11)$$

In the original algorithm, we need to save the streams of all pixel values of a $w \times w$ window in an array in order to find the local maximum in each cycle. In the second step of our proposed Kernel 1, we compare the value of the current stream with its corresponding window's maximum, and update the maximum when meeting the updating condition. For an input stream P_i , its corresponding window index j can be calculated according to Equation (12), where x_i and y_i are the coordinates of P_i . The local maximum of P_i 's corresponding window is denoted by $M(W_j)$. If P_i is larger than $M(W_j)$, then $M(W_j)$, $X(W_j)$, $Y(W_j)$ and $R(W_j)$ will be replaced by P_i , x_i , y_i and R_i ; Otherwise, $M(W_j)$, $X(W_j)$, $Y(W_j)$ and $R(W_j)$ will remain their previous values. After this step, we obtain the pixel value, the x and y coordinates, and the average radius of transect sampling for each window's local maximum. Consequently, we need four arrays to save and update the above four types of values. The size of each

array equals to the number of windows in the image. In each cycle, only the current stream need to be accessed and no offset is required for this step.

$$j = (y_i/W) \times (W/w) + (x_i\%w) \quad (12)$$

4.3. Kernel 2: Transect Sampling Radius Based Local Maximum Filtering

In the original algorithm described in Section 2, after obtaining the local maximum and the transect sampling radius of all windows, a circular-window-based local maximum filtering strategy is applied in order to find the new local maximum within the transect sampling radius. Similar to the previous phase, there are several problems if we map the original algorithm directly to FPGAs. On the one hand, only the result of one cycle in $w \times w$ cycles is valid and useful, while the results of the other $(w \times w - 1)$ cycles are ignored. On the other hand, in each cycle, it requires a large size of on-chip buffer to store the pixels of previous cycles in order to access all pixels within the circular window (with transect sampling radius). The value range of the required offset in each cycle is $[-n_t \times W - n_t, (w + n_t) \times W + (w + n_t)]$. In order to solve these problems, we reconstruct the original circular-window-based local maximum filtering so that the local maximum is searched across cycles instead of spaces. For our pipeline-friendly design of this phase, only the current stream needs to be accessed and no offset is required in each cycle.

Figure 5 shows the basic process of Kernel 2. In each cycle, we need to determine whether the current stream P_i can be the new local maximum of any window. For $j \in \{0, 1, \dots, N_w - 1\}$, the squared Euclidean distance between the coordinate of the current stream (P_i) and the coordinate of its corresponding window's local maximum ($M(W_j)$) can be calculated according to Equation (13). For the new local maximum of window W_j , its pixel value, x coordinate and y coordinate are denoted by $M'(W_j)$, $X'(W_j)$, and $Y'(W_j)$, respectively. If the value of P_i is larger than $M(W_j)$, and the squared Euclidean distance between the coordinates of P_i and $M(W_j)$ is smaller than or equal to the squared average radius of W_j , then $M'(W_j)$, $X'(W_j)$, and $Y'(W_j)$ will be replaced by P_i , x_i , and y_i ; Otherwise, $M'(W_j)$, $X'(W_j)$, and $Y'(W_j)$ will remain their previous values (i.e., $M(W_j)$, $X(W_j)$, and $Y(W_j)$). After this phase, we can obtain the coordinates of the updated local maximum of each window.

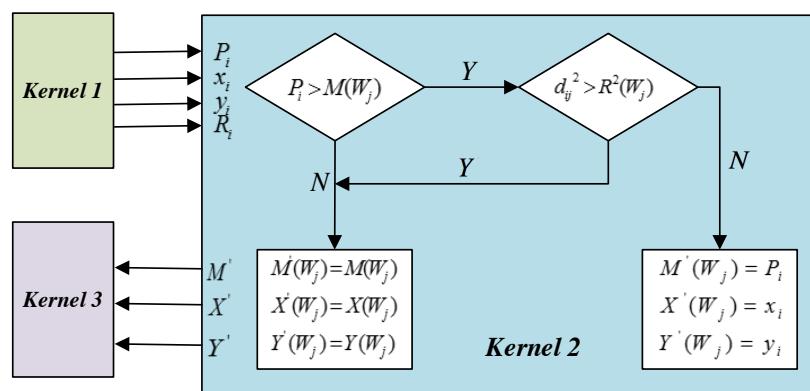


Figure 5. The basic process of Kernel 2.

$$d_{ij}^2 = (x_i - X(W_j))^2 + (y_i - Y(W_j))^2 \quad (13)$$

4.4. Kernel 3: Minimum Distance Filtering

In the last phase of the original tree crown detection algorithm, the minimum distance filtering is applied to the updated local maximum in order to merge the group of identified tree apexes' coordinates into one coordinate if their distance is smaller than a user-defined value, as shown in

Algorithm 2. Algorithm 3 defines the process of the improved minimum distance filtering of our proposed PF-TCD algorithm. The inputs of Kernel 3 are the updated local maximum's coordinates of each window obtained in Kernel 2. For each cycle, we need to calculate the Euclidean distance between the current input stream $(X'(W_j), Y'(W_j))$ and all updated local maximum's coordinates $(X'(W_{jj}), Y'(W_{jj}))$. MG_{jj} is used to record whether $(X'(W_{jj}), Y'(W_{jj}))$ should be merged with $(X'(W_j), Y'(W_j))$, in which values 1 and 0 indicate the two coordinates should or should not be merged. S_{jj} is used to record whether $(X'(W_{jj}), Y'(W_{jj}))$ is existing (S_{jj} equals True) or has been merged (S_{jj} equals False). Moreover, $(X''(W_{jj}), Y''(W_{jj}))$ is used to record all coordinates that should be merged with $(X'(W_j), Y'(W_j))$. N_{merge} denotes the total number of coordinates that should be merged with the current stream $(X'(W_j), Y'(W_j))$. X_{avg} and Y_{avg} denote the average coordinates of $(X''(W_{jj}), Y''(W_{jj}))$, which are the final tree crown detection results of our proposed PF-TCD algorithm.

Algorithm 3 Minimum distance filtering on FPGA

for all $jj \in \{0, 1, \dots, N_w - 1\}$ **do**

$$d_{jj} = dist((X'(W_j), Y'(W_j)), (X'(W_{jj}), Y'(W_{jj})))$$

if $S_j == True$ & $S_{jj} == True$ & $d_{jj} < d_{min}$ **then**

$$X''(W_{jj}) = X'(W_{jj}), Y''(W_{jj}) = Y'(W_{jj}), MG_{jj} = 1, S_{jj} = False$$

else

$$X''(W_{jj}) = 0, Y''(W_{jj}) = 0, MG_{jj} = 0$$

end if

end for

$$N_{merge} = \sum_{jj=0}^{N_t-1} MG_{jj}$$

$$X_{avg} = (N_{merge} \neq 0) ? \frac{\sum_{jj=0}^{N_t-1} X''(W_{jj})}{N_{merge}} : 0$$

$$Y_{avg} = (N_{merge} \neq 0) ? \frac{\sum_{jj=0}^{N_t-1} Y''(W_{jj})}{N_{merge}} : 0$$

To implement our proposed Algorithm 3 on FPGAs, we need to use an array in the length of N_W in order to record the $(X'(W_{jj}), Y'(W_{jj}))$ coordinate and the status S_{jj} for each window (for $jj \in 0, 1, \dots, N_W$). For each cycle, we need to read and update the value of S_{jj} according to whether $(X'(W_{jj}), Y'(W_{jj}))$ should be merged, which results in a loop dependency of 4 in our case. Consequently, we have to read a value every other four cycles and only 1/5 of the cycles produce useful results. In order to solve the above problem, we propose and implement a strategy for improving the design of Kernel 3 on FPGAs. We create an array in the length of $5 \times N_W$ so that the positions of reading and writing are different to avoid the loop dependency, as shown in Figure 6. The logic resource (LUTs and FFs) usage of this implementation is almost the same as using one array in the length of N_W . The only extra overhead is the space (i.e., BRAM) of the extra length of the array, which is only a small percentage of the whole resource usage. In this way, we do not need to use the idle cycles for array writing and the results can be generated in each cycle.

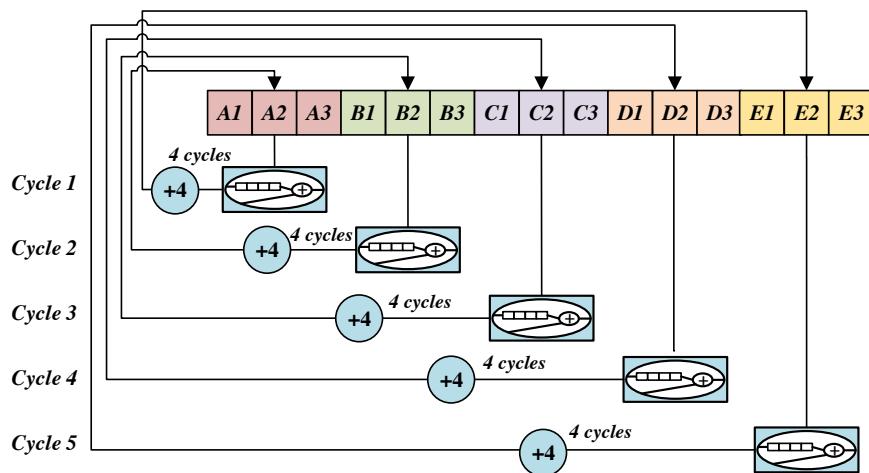


Figure 6. The proposed strategy of removing loop dependencies using extra BRAMs.

5. Experimental Results Analysis

In this section, we analyze the experiment results of our proposed PF-TCD algorithm on FPGAs. The hardware platform, the processing time and the speedup over original implementation will be described in Section 5.1. The resource utilization of each kernel will be described in Section 5.2. The tree crown detection results of our proposed approach will be analyzed in Section 5.3.

5.1. Performance Analysis of Our Proposed Approach

In this study, a Quickbird satellite image with a size of $12,188 \times 12,576$ pixels is used to evaluate the performance of our proposed approach. We implement the proposed PF-TCD algorithm using a high-level programming language called MaxJ (an extended version of Java). The kernel and manager code of PF-TCD algorithm is transformed into a hardware design using MaxCompiler and further processed into hardware code by FPGA vendor tools [40]. The MaxCompiler enables us to focus on the description of the hardware design by hiding the hardware details. The performance of PF-TCD algorithm is tested on a Maxeler MAX4 acceleration card with an Altera Stratix-V FPGA (running at 200 MHz) and 48GB DDR3 onboard memory.

We also compare the performance of our hardware implementation with a well-optimized software implementation of the original algorithm (implemented using multi-threaded C++, compiled with g++ with -O3 flags) on a 12-core CPU (Intel Xeon CPU E5-2697, running at 2.4 GHz). The running time (averaged from 10 repeated experiments) of the hardware solution and the software solution are 0.33 s and 6.12 s, indicating that our proposed PF-TCD algorithm on FPGAs achieves the speedup of 18.75 times over the original algorithm on 12-core CPU and meets the real-time data processing requirements of satellites.

5.2. Resource Utilization of Each Kernel

In our proposed approach, the image size of each block is set as 116×116 pixels, in which the width of the overlap is 16 pixels (similar to the crown diameter of a mature oil palm tree). The window width (w) of 5, 10 and 15 are evaluated in this study, which should be smaller than the diameter of a tree crown. When the w is set as 10, the average F1-score of the six selected regions is over 8% higher than those obtained from other two cases. We also evaluate the number of pixels in a transect (n_t) from 8 to 12, which should be similar to or larger than the radius of a tree crown. The difference of F1-score obtained from each case is smaller than 1% and the image processing time increases with n_t . Consequently, the parameter w is set as 10 and the parameter n_t is set as 8 for our study area. The parameter d_{min} (the minimum allowed distance between two tree apexes) is set as 5, which

guarantees that the coordinates corresponding to the same tree can be averaged into one coordinate properly. As the value range of our data is 0 to 255, 8-bit fixed point computations are used for this algorithm. For our proposed PF-TCD algorithm, the accumulative number of cycles of Kernel 1, Kernel 2, and Kernel 3 (denoted by N_{Cycle_K1} , N_{Cycle_K2} , N_{Cycle_K3}) can be calculated according to Formula (14), of which the total number of cycles (denoted by N_{Cycle_Total}) equals the accumulative number of cycles of Kernel 3. Kernel 1 and Kernel 2 are fully-pipelined and the input of Kernel 2 is the output of Kernel 1. For Kernel 3, the incremental number of cycles equals to two times of the number of windows, which takes only a small percentage of the total number of cycles (1.5%).

$$\begin{aligned} N_{Cycle_K1} &= (W + 2 \times n_t)^2 \\ N_{Cycle_K2} &= N_{Cycle_K1} + (W/w) + 2 \\ N_{Cycle_Total} &= N_{Cycle_K3} = N_{Cycle_K2} + 2 \times (W/w)^2 \end{aligned} \quad (14)$$

Table 1 summarizes the resource utilization of an image block in each main phase. The total resource utilization of an image block on the Stratix-5SGSD8 FPGA is 28.33% of the LUTs, 27.82% of the FFs, 25.47% of the DSPs and 29.61% of the BRAMs. Based on this resource utilization, three image blocks in 116×116 pixels can be processed in parallel in our proposed approach. The power consumption of our proposed approach on FPGAs is 20–25 W, which is significantly lower than the power consumption required by GPUs and CPUs.

Table 1. The resource utilization of our proposed PF-TCD algorithm on FPGAs.

Resource	Kernel 1	Kernel 2	Kernel 3	Manager	Total
LUTs	1636	31,229	2852	38,621	74,338 (28.33%)
FFs	2847	50,062	6250	86,842	146,001 (27.82%)
DSPs	0	370	130	0	500 (25.47%)
BRAMs	0	304	1	455	760 (29.61%)

5.3. The Detection Results of Our Proposed Approach

To evaluate the detection results of our proposed approach, we compare the predicted tree crown coordinates of each region with the ground truth collected through manually labeling by specialist. The calculating processes of the precision, recall, and F1-score are described from Equations (15)–(17), in which TP denotes the true positive detection indicating the number of correctly detected oil palm trees, FP denotes the false positive detection indicating the number of other objects detected as oil palm trees, and FN denotes the false negative detection indicating the number of oil palm trees not detected. Similar to [4], if the Euclidean spatial distance between the coordinates of a detected oil palm tree and a ground truth oil palm tree is smaller than or equal to 5 pixels, then the oil palm tree is considered as detected correctly.

$$Precision = \frac{TP}{TP + FP} \quad (15)$$

$$Recall = \frac{TP}{TP + FN} \quad (16)$$

$$F1-score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (17)$$

Figures 7 and 8 show the detection image results of six selected regions obtained from our proposed PF-TCD algorithm and the original algorithm, in which the red points, green squares and blue squares indicate the correctly detected oil palm trees (TP), the oil palm trees not detected (FN), and other objects detected as oil palms (FP), respectively. Table 2 shows the detection results of each region obtained from our proposed PF-TCD algorithm and the original algorithm, in terms of TP, FP,

FN, Precision, Recall and F1-score. The beginning (x, y) coordinates (the pixel offset to the top-left corner of the whole study area) of six regions are (3896, 1715), (5390, 2733), (8783, 3768), (10,708, 6519), (8210, 8678) and (1743, 11,384) in pixels. The sizes of six regions are 600×600 pixels. Experiment results demonstrate that the detection F1-scores obtained from our proposed PF-TCD algorithm are between 83.98% and 92.15% for six regions. The difference between the F1-scores obtained from PF-TCD algorithm on FPGAs and those obtained from the original algorithm on CPU is smaller than 1%, even though we use special angles (such as 45° and 90°) for transact sampling and fixed point computations in PF-TCD algorithm.

Table 2. The oil palm tree detection results of each region.

Method	Index	Region 1	Region 2	Region 3	Region 4	Region 5	Region 6
PF-TCD algorithm	TP	1033	1323	739	684	986	1380
	FP	206	196	121	164	104	259
	FN	72	104	43	97	64	125
	Precision	83.37%	87.10%	85.93%	80.66%	90.46%	84.20%
	Recall	93.48%	92.71%	94.50%	87.58%	93.90%	91.69%
	F1-score	88.14%	89.82%	90.01%	83.98%	92.15%	87.79%
Original algorithm	TP	1025	1320	741	681	980	1382
	FP	212	200	119	165	110	257
	FN	80	106	41	100	70	123
	Precision	82.86%	86.84%	86.16%	80.50%	89.91%	84.32%
	Recall	92.76%	92.57%	94.76%	87.20%	93.33%	91.83%
	F1-score	87.53%	89.61%	90.26%	83.71%	91.59%	87.91%

6. Discussion

In this section, we analyze the size of image blocks and the image processing time of our proposed approach. The impact of the size of each image block on the performance of our proposed approach will be analyzed in Section 6.1. The running time for processing images in different sizes will be analyzed in Section 6.2.

6.1. The Size of Image Blocks

In the blocking module of our proposed approach, the size of each image block has a great influence on the performance of our proposed approach. When the size of each image block is getting smaller, the resource utilization of each block will be less and more image blocks can be processed in parallel. However, as the image block contains overlaps with their neighbors to guarantee that each tree crown can be completely included in at least one image block, smaller image blocks will increase the total number of image blocks and duplicated pixels. In contrast, when the size of each image block is getting larger, the resource utilization of each block will be larger and fewer image blocks can be processed in parallel. However, the number of duplicated pixels to be processed will be reduced and fewer cycles will be required to process the whole image. After tuning the size of each image block, we set the size of image block as 116×116 to maximize the overall performance.

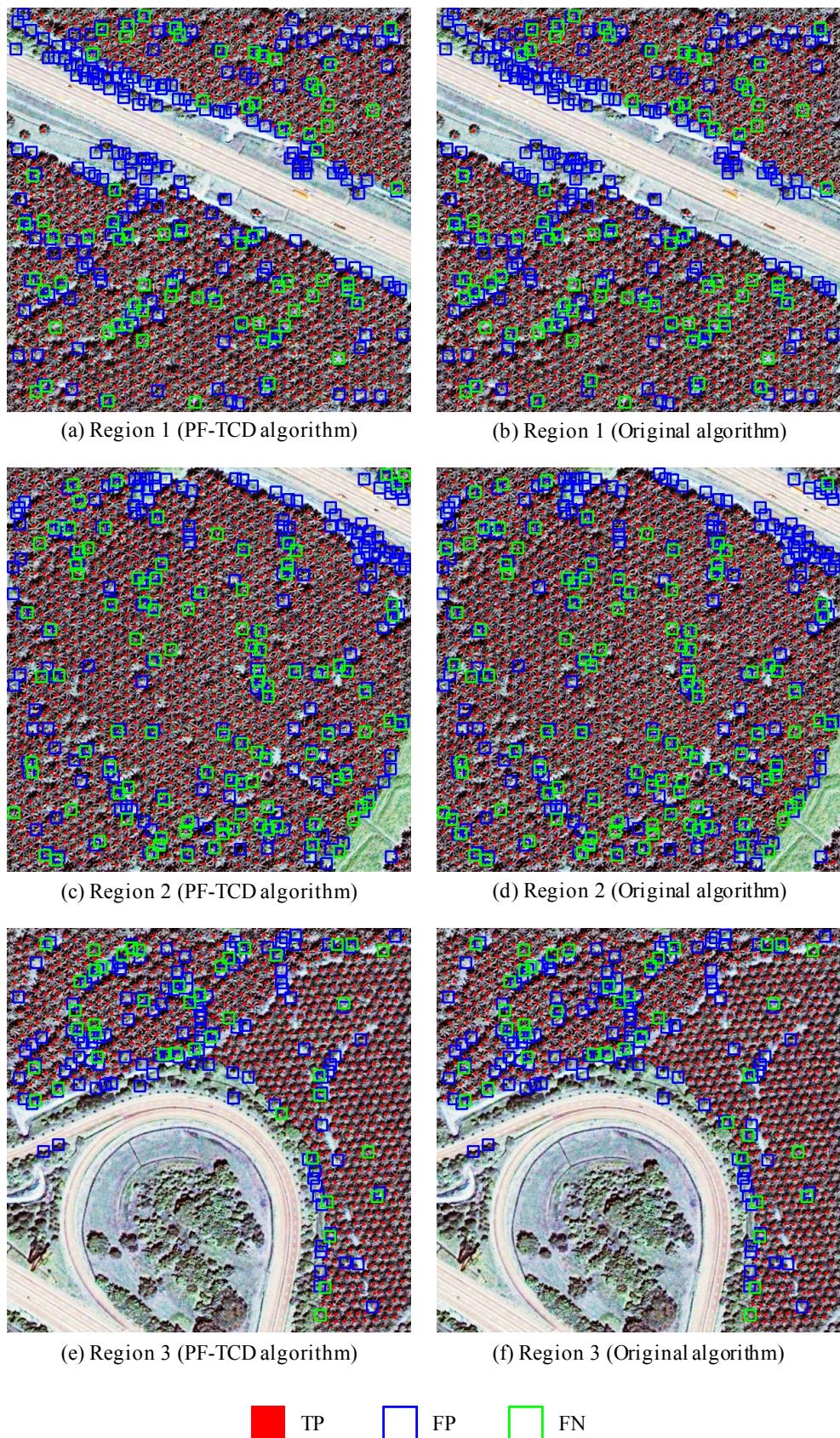


Figure 7. The detection image results of region 1, 2 and 3 obtained from our proposed PF-TCD algorithm (left) and the original algorithm (right).

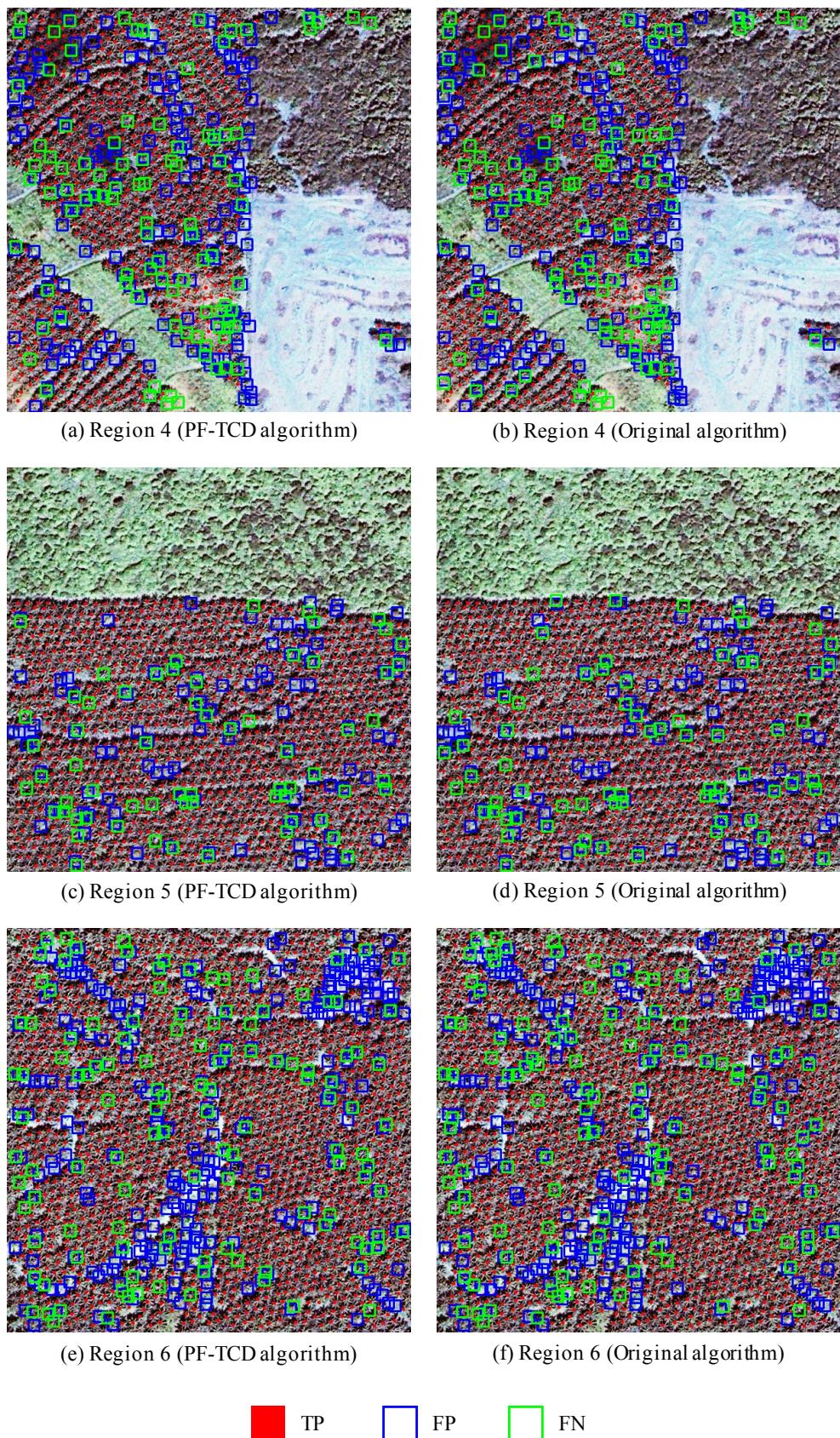


Figure 8. The detection image results of region 4, 5 and 6 obtained from our proposed PF-TCD algorithm (left) and the original algorithm (right).

6.2. The Running Time for Processing Images in Different Sizes

Figure 9 shows the running time of our proposed approach for processing remote sensing images in different sizes. We can find that the running time increases almost linearly with the image size. This is because when the FPGA resource utilization and the size of each image block are fixed, the number of image blocks will increase with the size of the remote sensing image. The total number of cycles for processing an image increases linearly with the image size, resulting in the linear increasing of the running time. Based on this linear relationship, we can further predict the running time for processing larger remote sensing images.

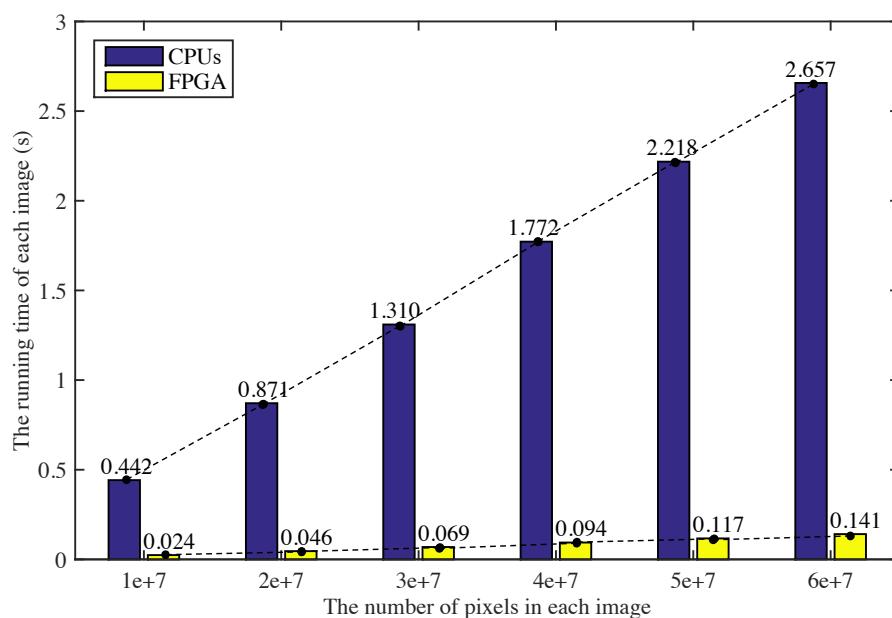


Figure 9. The running time for processing images in different sizes.

7. Conclusions

In this research, we propose an FPGA-based approach for large-scale remote sensing image processing in real time. It is a complete high performance solution for automatic tree crown detection from the raw satellite images to the final detection results. To satisfy the power, latency and performance requirements of the practical scenarios of on-board data processing, we reconstruct and modify the workflow of the original algorithm and designed a pipelined-friendly tree crown detection algorithm (PF-TCD) with three computational kernels. We optimize and adjust the local maximum filtering, transact sampling, and minimum distance filtering algorithms of the tree crown detection approach to reduce the resource utilization and the idle cycles, and balance the utilization of different resources (LUTs, FFs, DSPs, and BRAMs) to avoid any of them becoming the performance bottleneck.

Compared with the well-optimized software implementation of the original algorithm on an Intel 12-core CPU, our proposed PF-TCD algorithm obtains the speedup of 18.75 times for a large-scale remote sensing image with $12,188 \times 12,576$ pixels. The image processing time is only 0.33 s, which satisfies the requirements of real-time data processing on satellites. The detection F1-scores obtained from our proposed PF-TCD algorithm are between 83.98% and 92.15% for six selected regions. In our future work, we plan to further optimize our proposed approach and apply it to larger-scale datasets. We will also explore the potential of reconfigurable platforms and develop high performance solutions for other remote sensing image analysis tasks.

Author Contributions: Conceptualization, W.L. (Weijia Li), C.H., H.F. and W.L. (Wayne Luk); Data curation, W.L. (Weijia Li) and L.Y.; Formal analysis, W.L. (Weijia Li); Funding acquisition, H.F.; Investigation, W.L. (Weijia Li) and C.H.; Methodology, W.L. (Weijia Li) and C.H.; Project administration, H.F.; Resources, H.F. and L.Y.; Software, W.L. (Weijia Li) and C.H.; Supervision, H.F. and W.L. (Wayne Luk); Validation, W.L. (Weijia Li), C.H. and J.Z.; Visualization, W.L. (Weijia Li), C.H. and J.Z.; Writing—original draft, W.L. (Weijia Li) and C.H.; Writing—review & editing, H.F., R.D., M.X. and W.L. (Wayne Luk).

Funding: This research was supported in part by the National Key R&D Program of China (Grant No. 2017YFA0604500 and 2017YFA0604401), by the National Natural Science Foundation of China (Grant No. 51761135015), and by the Center for High Performance Computing and System Simulation, Pilot National Laboratory for Marine Science and Technology (Qingdao), the European Union Horizon 2020 Research and Innovation Programme under grant agreement number 671653, UK EPSRC (EP/I012036/1, EP/L00058X/1, EP/L016796/1 and EP/N031768/1), and Maxeler and Intel Programmable Solutions Group.

Acknowledgments: The authors would like to thank the editors and reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, W.; He, C.; Fu, H.; Luk, W. An FPGA-based tree crown detection approach for remote sensing images. In Proceedings of the IEEE 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, Australia, 11–13 December 2017; pp. 231–234.
2. Pouliot, D.; King, D.; Bell, F.; Pitt, D. Automated tree crown detection and delineation in high-resolution digital camera imagery of coniferous forest regeneration. *Remote Sens. Environ.* **2002**, *82*, 322–334. [\[CrossRef\]](#)
3. Tan, K.P.; Kanniah, K.D.; Cracknell, A.P. Use of UK-DMC 2 and ALOS PALSAR for studying the age of oil palm trees in southern peninsular Malaysia. *Int. J. Remote Sens.* **2013**, *34*, 7424–7446. [\[CrossRef\]](#)
4. Li, W.; Fu, H.; Yu, L.; Cracknell, A. Deep Learning Based Oil Palm Tree Detection and Counting for High-Resolution Remote Sensing Images. *Remote Sens.* **2016**, *9*, 22. [\[CrossRef\]](#)
5. Chirici, G.; Giuliarelli, D.; Biscontini, D.; Tonti, D.; Mattioli, W.; Marchetti, M.; Corona, P. Large-scale monitoring of coppice forest clearcuts by multitemporal very high resolution satellite imagery. A case study from central Italy. *Remote Sens. Environ.* **2011**, *115*, 1025–1033. [\[CrossRef\]](#)
6. Li, W.; He, C.; Fang, J.; Zheng, J.; Fu, H.; Yu, L. Semantic Segmentation-Based Building Footprint Extraction Using Very High-Resolution Satellite Images and Multi-Source GIS Data. *Remote Sens.* **2019**, *11*, 403. [\[CrossRef\]](#)
7. Ke, Y.H.; Zhang, W.H.; Quackenbush, L.J. Active contour and hill climbing for tree crown detection and delineation. *Photogramm. Eng. Remote Sens.* **2010**, *76*, 1169–1181. [\[CrossRef\]](#)
8. Santos, A.M.D.; Mitja, D.; Delaître, E.; Demagistri, L.; Miranda, I.D.S.; Libourel, T.; Petit, M. Estimating babassu palm density using automatic palm tree detection with very high spatial resolution satellite images. *J. Environ. Manag.* **2017**, *193*, 40–51. [\[CrossRef\]](#) [\[PubMed\]](#)
9. Ke, Y.; Quackenbush, L.J. A review of methods for automatic individual tree-crown detection and delineation from passive remote sensing. *Int. J. Remote Sens.* **2011**, *32*, 4725–4747. [\[CrossRef\]](#)
10. Quackenbush, L.J.; Hopkins, P.F.; Kinn, G.J. Using template correlation to identify individual trees in high resolution imagery. In Proceedings of the American Society for Photogrammetry & Remote Sensing (ASPRS) 2000 Annual Conference Proceedings, Washington, DC, USA, 22–26 May 2000.
11. Leckie, D.G.; Gougeon, F.A.; Tinis, S.; Nelson, T.; Burnett, C.N.; Paradine, D. Automated tree recognition in old growth conifer stands with high resolution digital imagery. *Remote Sens. Environ.* **2005**, *94*, 311–326. [\[CrossRef\]](#)
12. López-López, M.; Calderón, R.; González-Dugo, V.; Zarco-Tejada, P.J.; Fereres, E. Early Detection and Quantification of Almond Red Leaf Blotch Using High-Resolution Hyperspectral and Thermal Imagery. *Remote Sens.* **2016**, *8*, 276. [\[CrossRef\]](#)
13. Malek, S.; Bazi, Y.; Alajlan, N.; AlHichri, H.; Melgani, F. Efficient framework for palm tree detection in UAV images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 4692–4703. [\[CrossRef\]](#)
14. Nevalainen, O.; Honkavaara, E.; Tuominen, S.; Viljanen, N.; Hakala, T.; Yu, X.; Hyppä, J.; Saari, H.; Pöölönen, I.; Imai, N.; et al. Individual tree detection and classification with UAV-based photogrammetric point clouds and hyperspectral imaging. *Remote Sens.* **2017**, *9*, 185. [\[CrossRef\]](#)

15. Guirado, E.; Tabik, S.; Alcaraz-Segura, D.; Cabello, J.; Herrera, F. Deep-learning versus OBIA for scattered shrub detection with Google earth imagery: Ziziphus Lotus as case study. *Remote Sens.* **2017**, *9*, 1220.
16. Li, W.; Dong, R.; Fu, H.; Yu, L. Large-Scale Oil Palm Tree Detection from High-Resolution Satellite Images Using Two-Stage Convolutional Neural Networks. *Remote Sens.* **2019**, *11*, 11.
17. Freudenberg, M.; Nölke, N.; Agostini, A.; Urban, K.; Wörgötter, F.; Kleinn, C. Large Scale Palm Tree Detection in High Resolution Satellite Images Using U-Net. *Remote Sens.* **2019**, *11*, 312. [\[CrossRef\]](#)
18. Li, W.; Fu, H.; Yu, L.; Gong, P.; Feng, D.; Li, C.; Clinton, N. Stacked Autoencoder-based deep learning for remote-sensing image classification: A case study of African land-cover mapping. *Int. J. Remote Sens.* **2016**, *37*, 5632–5646. [\[CrossRef\]](#)
19. Santos, L.; Berrojo, L.; Moreno, J.; López, J.F.; Sarmiento, R. Multispectral and hyperspectral lossless compressor for space applications (HyLoC): A low-complexity FPGA implementation of the CCSDS 123 standard. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 757–770.
20. Bing, Z.; Wei, Y.; Gao, L.; Chen, D. Real-time target detection in hyperspectral images based on spatial-spectral information extraction. *Eur. J. Adv. Signal Process.* **2012**, *2012*, 142.
21. Liu, D.; Zhou, G.; Huang, J.; Zhang, R.; Shu, L.; Zhou, X.; Xin, C.S. On-Board Georeferencing Using FPGA-Based Optimized Second-Order Polynomial Equation. *Remote Sens.* **2019**, *11*, 124.
22. Bernabe, S.; Sanchez, S.; Plaza, A.; Lopez, S.; Benediktsson, J.A.; Sarmiento, R. Hyperspectral unmixing on GPUs and multi-core processors: A comparison. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 1386–1398.
23. Li, W.; Fu, H.; You, Y.; Yu, L.; Fang, J. Parallel Multiclass Support Vector Machine for Remote Sensing Data Classification on Multicore and Many-Core Architectures. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 4387–4398. [\[CrossRef\]](#)
24. Wu, Z.; Liu, J.; Plaza, A.; Li, J.; Wei, Z. GPU implementation of composite kernels for hyperspectral image classification. *IEEE Geosci. Remote Sens. Lett.* **2015**, *12*, 1973–1977.
25. González, C.; Bernabé, S.; Mozos, D.; Plaza, A. FPGA implementation of an algorithm for automatically detecting targets in remotely sensed hyperspectral images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 4334–4343. [\[CrossRef\]](#)
26. Mohammadnia, M.R.; Shannon, L. A multi-beam Scan Mode Synthetic Aperture Radar processor suitable for satellite operation. In Proceedings of the 2016 IEEE 27th International Conference on IEEE Application-Specific Systems, Architectures and Processors (ASAP), London, UK, 6–8 July 2016; pp. 83–90.
27. Lee, C.A.; Gasster, S.D.; Plaza, A.; Chang, C.I.; Huang, B. Recent developments in high performance computing for remote sensing: A review. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2011**, *4*, 508–527. [\[CrossRef\]](#)
28. Wang, S.; Niu, X.; Ma, N.; Luk, W.; Leong, P.; Peng, Y. A Scalable Dataflow Accelerator for Real Time Onboard Hyperspectral Image Classification. In *International Symposium on Applied Reconfigurable Computing*; Springer: Berlin, Germany, 2016; pp. 105–116.
29. Gonzalez, C.; Lopez, S.; Mozos, D.; Sarmiento, R. A novel FPGA-based architecture for the estimation of the virtual dimensionality in remotely sensed hyperspectral images. *J. Real-Time Image Process.* **2015**, *43*, 1–12. [\[CrossRef\]](#)
30. Shan, N.; Wang, X.S.; Wang, Z.S. Efficient FPGA implementation of cloud detection for real-time remote sensing image processing. In Proceedings of the 2010 Conference on Postgraduate Research in IEEE Asia Pacific Microelectronics and Electronics (PrimeAsia), Shanghai, China, 22–24 September 2010; pp. 190–193.
31. Tang, J.W.; Shaikh-Husin, N.; Sheikh, U.U.; Marsono, M.N. FPGA-based real-time moving target detection system for unmanned aerial vehicle application. *Int. J. Reconfigur. Comput.* **2016**, *2016*. [\[CrossRef\]](#)
32. Chen, Y.L.; Wu, C.C.; Lin, H.C.; Lin, C. A parallel approach of multi-level morphological active contour algorithm for individual tree detection and crown delineation. In Proceedings of the 2013 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Melbourne, Australia, 21–26 July 2013; pp. 2601–2604.
33. Duncanson, L.; Cook, B.; Hurt, G.; Dubayah, R. An efficient, multi-layered crown delineation algorithm for mapping individual tree structure across multiple ecosystems. *Remote Sens. Environ.* **2014**, *154*, 378–386. [\[CrossRef\]](#)
34. Jiang, H.; Chen, S.; Li, D.; Wang, C.; Yang, J. Papaya Tree Detection with UAV Images Using a GPU-Accelerated Scale-Space Filtering Method. *Remote Sens.* **2017**, *9*, 721. [\[CrossRef\]](#)

35. Srestasathiern, P.; Rakwatin, P. Oil palm tree detection with high resolution multi-spectral satellite imagery. *Remote Sens.* **2014**, *6*, 9749–9774. [[CrossRef](#)]
36. Cheng, Y.; Le, Y.; Xu, Y.; Hui, L.; Cracknell, A.P.; Kanniah, K.; Peng, G. Mapping oil palm extent in Malaysia using ALOS-2 PALSAR-2 data. *Int. J. Remote Sens.* **2018**, *39*, 432–452. [[CrossRef](#)]
37. Cheng, Y.; Le, Y.; Xu, Y.; Liu, X.; Hui, L.; Cracknell, A.P.; Kanniah, K.; Peng, G. Towards global oil palm plantation mapping using remote-sensing data. *Int. J. Remote Sens.* **2018**, *39*, 5891–5906. [[CrossRef](#)]
38. Malaysian Oil Palm Plantation Dataset. Available online: https://github.com/dongrunmin/oil_palm_data/blob/master/segmentation_map (accessed on 6 March 2019).
39. Lachowicz, S.; Pfleiderer, H.J. Fast evaluation of nonlinear functions using FPGAs. *Adv. Radio Sci.* **2008**, *6*, 233–237. [[CrossRef](#)]
40. Summers, S.; Rose, A.; Sanders, P. Using MaxCompiler for the high level synthesis of trigger algorithms. *J. Instrum.* **2017**, *12*, C02015. [[CrossRef](#)]



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).